

物件導向三特性

三個特性

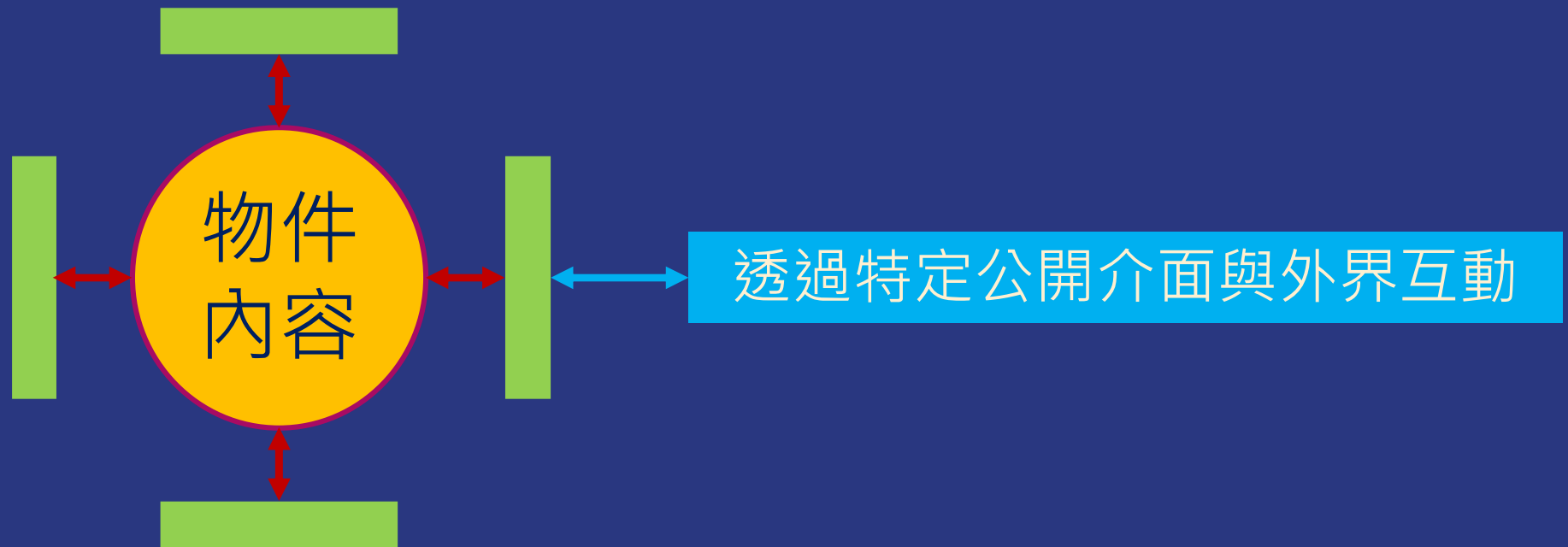
- 繼承
- 封裝
- 多型

繼承

- 繼承者會擁有被繼承者的型別特徵（非靜態）
- C# 中的繼承
 - 繼承一個上層類別（只能一個）
 - 實做介面（可以多個）

封裝

- 隱藏不必要為外界所知的資訊
- 隱藏行為的變化



多型

- 廣義多型 (universal polymorphism)
 - 繼承式多型 (inclusion)
 - 參數式多型 (parametric)
- 特設多型 (ad hoc polymorphism)
 - 多載 (overloading)
 - 強制同型 (coercions)

來源: [On Understanding Types, Data Abstraction, and Polymorphism](#)
1.3. Kinds of Polymorphism

先來談對於型別的存取修飾詞

- 在命名空間中宣告的型別別可為 `public` 或 `internal`（預設，你沒有寫的時候叫預設）

類別內部成員的存取修飾詞

- 類別內的成員可以宣告為
 - `private` (預設)
 - `internal`
 - `protected`
 - `protected internal`
 - `public`

類別特性的修飾詞

- **abstract**

- 類別宣告為 **abstract** 表示抽象類別，不具有公開建構式（預設建構式為 **protected**，而且無法使用 **public** 或 **internal**，宣告會過，但一遇到 **new** 就掛點了）。這表示你不能直接以 **new** 關鍵字建立其執行個體。
- 抽象類別通常是用來當作父類別使用。
- 抽象類別可以擁有實做不完整的抽象成員。

- **sealed**

- 類別宣告為 **sealed** 表示為密封類別，也就是這個類別無法作為其他類別的父類別；換句話說，沒有任何一個類別可以繼承自被宣告為 **sealed** 的類別。

類別內部成員的特性修飾詞

- **abstract**

- 成員以 **abstract** 宣告表示為抽象成員，亦即這個成員不具備完整實作，而繼承此成員所在抽象類別的子類別必須實作此成員（除非子類別也被宣告為抽象類別，並且宣告此成員為抽象成員）。

- **virtual**

- 成員以 **virtual** 宣告表示為虛擬成員，而繼承此成員所在類別的子類別可以覆寫(override) 此成員

類別內部成員的特性修飾詞

- **override**

- 表示覆寫父類別成員，當此成員所在類別的父類別中是以 **abstract**，**virtual** 或 **override** 宣告的時候，可以宣告 **override** 覆寫此成員

- **sealed**

- 表示密封此成員，一定和 **override** 同時使用，表示從此以後這個成員就無法再被覆寫了

- **new**

- 表示遮蔽或隱藏成員，**99.9%** 不會用它，它是個危險的東西。
- 參考：MSDN new 修飾詞 (C# 參考)
- <https://msdn.microsoft.com/zh-tw/library/435f1dw2.aspx>

LAB

抽象類別與繼承的實作

新增一個方案及專案

- 方案名稱：AllClassSamples
- 專案名稱：AllClassSample001
- 範本：Console Application

加入 MyShape Class，並建立其內容

```
/// <summary>
/// 宣告一個抽象類別
/// </summary>
public abstract class MyShape
{
    /// <summary>
    /// 宣告一個抽象方法
    /// </summary>
    /// <returns></returns>
    public abstract double GetArea();
}
```

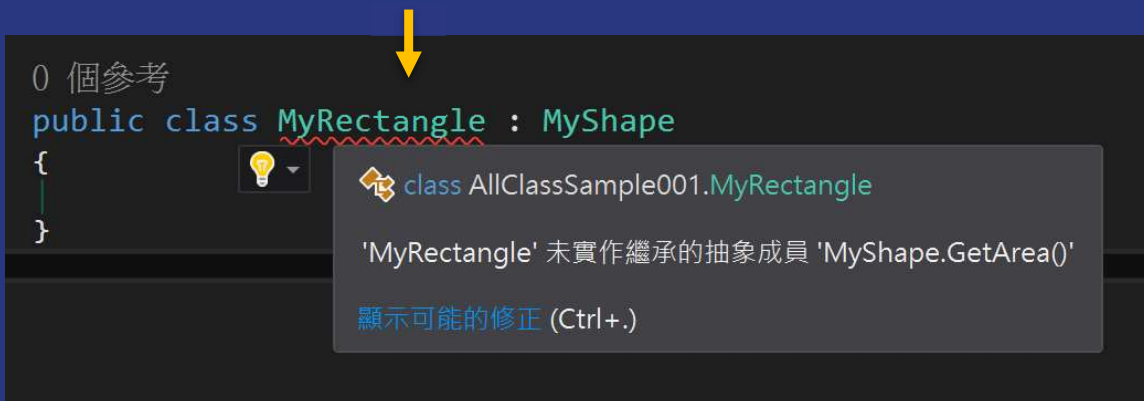
為什麼 MyShape 是抽象的

- **MyShape** 代表一個形狀的抽象，而我們目前不知道這個形狀會是方形、三角形還是圓形。所以他只有一個抽象的定義。
- 所謂形狀，就是一種封閉的平面區域，封閉平面區域必然具有面積。
- 因此，這個 **MyShape** 定義了一個抽象的方法叫 **GetArea**，子類別繼承時，會因為形狀的不同而實作不同方式取得面積的程式碼。

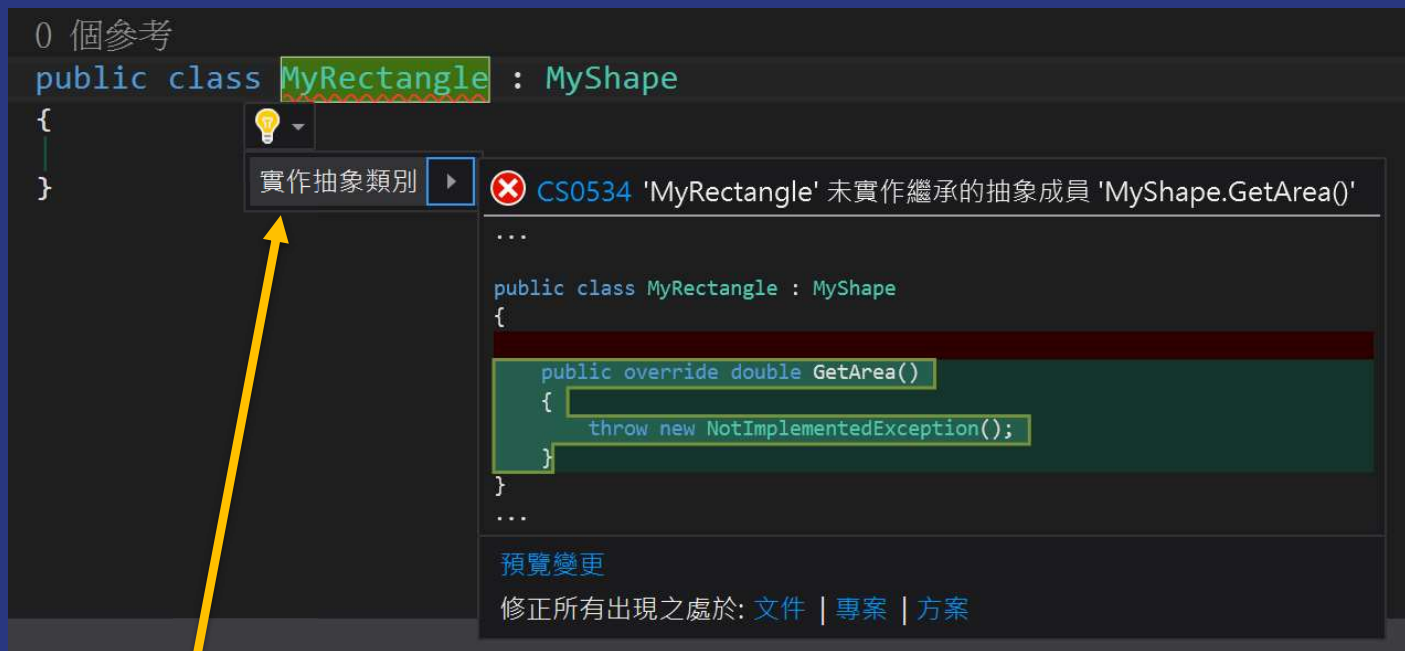
繼承 MyShape 加入 MyRectangle Class 先在此暫停一下

```
public class MyRectangle : MyShape  
{  
  
}
```

你會看到 MyRectangle 上有個紅色的波浪警示



前述的紅色波浪警示表示 `MyRectangle` 必須要實作 `MyShape` 的抽象方法（也就是 `GetArea`）但你沒有實作。
此時，請按下燈泡



Visual Studio 提示你應該要【實作抽象類別】
請按下它

Visual Studio 會幫你完成以下的程式碼
(有看到 `override` 嗎？表示 `MyRectangle` 覆寫了 `MyShape` 的 `GetArea` 方法)

```
public class MyRectangle : MyShape
{
    public override double GetArea()
    {
        throw new NotImplementedException();
    }
}
```

注意：這個 `throw new NotImplementedException()` 表示程式執行到此會拋出一個 `NotImplemented`（代表沒有實作完成）的例外，所以要把這行拿掉。

```
public class MyRectangle : MyShape
{
    public override double GetArea()
    {
    }
}
```

完成整個 MyRectangle 的內容

```
public class MyRectangle : MyShape
{
    public double Width { get; set; }
    public double Height { get; set; }
    public override double GetArea()
    {
        return Width * Height;
    }
}
```

然後依樣畫葫蘆來做的圓形的類別
請不要完全用打字的，善用 Visual Studio 的功能

```
public class MyCircle : MyShape
{
    public double Radius { get; set; }
    public override double GetArea()
    {
        return Math.PI * Math.Pow(Radius, 2);
    }
}
```

這裡用到了 **Math Class**，這個類別裡面有許多專門用來
做數學計算的方法和欄位

- (1) **Math.PI** 是一個唯讀欄位，代表 π
- (2) **Math.Pow** 是一個靜態方法，代表乘冪運算（就是 x 的 y 次方這種）

回到 Program class , 修改 Main method

```
static void Main(string[] args)
{
    MyShape rect = new MyRectangle() { Width = 10, Height = 10 };
    Console.WriteLine($"方形的面積是 {rect.GetArea()}");
    MyShape circle = new MyCircle() { Radius = 3 };
    Console.WriteLine($"圓形的面積是 {circle.GetArea()}");
    Console.ReadLine();
}
```

各位是否有注意到，雖然變數的型別是 **MyShape**，但是每個執行個體很盡責的呼叫了自己的覆寫後的 **GetArea** 這是一件非常重要的事情。

執行

注意

- 雖然抽象類別可以擁有抽象的成員，但未必每個成員都需要宣告成抽象的。它也可以擁有一般的成員。
- 欄位沒有抽象這回事。
- 你可以試著寫 `MyShape shape = new MyShape();` 看看會發生甚麼事。

討論

- 抽象類別在設計上佔有很重要的地位，請互相討論你們對於剛剛的範例程式的內容。
- `new MyShape();` 為何行不通？



virtual member

- 虛擬成員和抽象成員不同的是虛擬成員是具備有完整實作的內容，也就是具備完整的 { } 區段，即使內容為空都視為有實作。
- 虛擬成員和抽象成員不同的第二點是繼承者可以選擇覆寫或不覆寫這個成員，不像抽象成員一定要被覆寫。

LAB

虛擬成員與繼承的覆寫

在 AllClassSamples 加入新專案

- 方案名稱：AllClassSamples
- 專案名稱：AllClassSample002
- 範本：Console Application

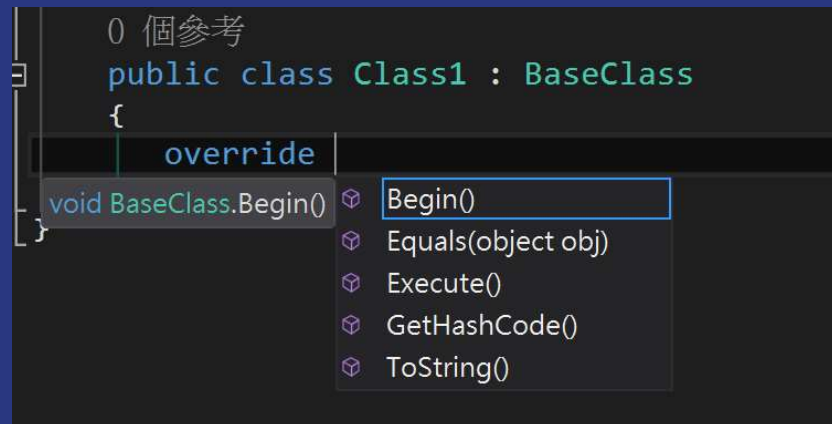
先建立 BaseClass class
兩個方法都宣告為 virtual (虛擬)

```
public class BaseClass
{
    public virtual void Execute()
    {
        Console.WriteLine("BaseClass Execute Method");
    }

    public virtual void Begin()
    {
        Console.WriteLine("BaseClass Begin Method");
    }
}
```

覆寫成員的方式

- (1) 輸入 `override`
- (2) 然後按下空白鍵
- (3) 就會出現可以覆寫的成員選單



建立 Class1 class (繼承 BaseClass)
並 override (覆寫) Execute()

```
public class Class1 : BaseClass
{
    public override void Execute()
    {
        Console.WriteLine("Class1 Execute Method");
    }
}
```

回到 Program class , 修改 Main method

```
static void Main(string[] args)
{
    BaseClass o1 = new Class1();
    o1.Execute();
    o1.Begin();

    Console.ReadLine();
}
```

執行時請觀察 `o1.Execute()` 和 `o1.Begin()` 的結果

執行

- (1)接著同一個專案繼續，建立 Class2 class(繼承 Class1)
- (2)override (覆寫) Execute()且宣告為 sealed (密封)
- (3)Override Begin()

```
public class Class2 : Class1
{
    public override sealed void Execute()
    {
        Console.WriteLine("Class2 Execute Method");
    }

    public override void Begin()
    {
        Console.WriteLine("Class2 Begin Method");
    }
}
```

- (1)這表示父類別(Class1)的方法(Execute())若為 override，子類別(Class2)一樣可以覆寫它
- (2)若祖父類別(BaseClass)的方法(Begin())若為 virtual/override，而父類別(Class1)雖然沒有覆寫，但子類別(Class2)一樣可以覆寫

回到 Program class，修改 Main method

```
static void Main(string[] args)
{
    BaseClass o1 = new Class1();
    o1.Execute();
    o1.Begin();

    BaseClass o2 = new Class2();
    o2.Execute();
    o2.Begin();

    Console.ReadLine();
}
```

執行時請觀察 `o2.Execute()` 和 `o2.Begin()` 的結果
並與前面的結果比較

執行

- (1) 接著同一個專案繼續，建立 Class3 class(繼承 Class2)
- (2) 試圖硬要覆寫被 sealed 的 Execute
- (3) 會出現如下圖的錯誤

```
0 個參考
public class Class3 : Class2
{
    5 個參考
    public override void Execute()
    {
        Console.WriteLine("Class3")
    }
}
```

void Class3.Execute()
'Class3.Execute()': 無法覆寫繼承的成員 'Class2.Execute()'，因為其已密封。

方法多載 (overloading)

- 同樣的方法名稱，不同的參數清單
 - 型別不同 或 參數數量不同
- 不可以單純只多載回傳值
- 又稱為【同名異式】

合法的多載

```
static int Add(int x, int y)
{ return x + y; }
```

```
static int Add(int x, int y, int z)
{ return x + y + z; }
```

```
static double Add(double x, double y)
{ return x + y; }
```

```
static string Add(string x, string y)
{ return x + y; }
```

```
static string Add(string x, int y)
{ return x + "整數" + y.ToString(); }
```

不合法的多載

```
static void Add()  
{  
}
```

```
static int Add()  
{  
}
```

```
static void Sub(int x)  
{  
}
```

```
static int Sub(int y)  
{  
}
```

LAB

方法多載

新增一個方案及專案

- 方案名稱：MethodOverloadingSamples
- 專案名稱：MethodOverloadingSample001
- 範本：Console Application

在 Program Class 加入以下的多載方法

```
static int Add(int x, int y)
{ return x + y; }
```

```
static int Add(int x, int y, int z)
{ return x + y + z; }
```

```
static double Add(double x, double y)
{ return x + y; }
```

```
static string Add(string x, string y)
{ return x + y; }
```

```
static string Add(string x, int y)
{ return x + "整數" + y.ToString(); }
```

直接在 Main Method 加入 程式碼

```
static void Main(string[] args)
{
    Console.WriteLine(Add(1, 1));
    Console.WriteLine(Add(1, 2, 3));
    Console.WriteLine(Add(1.5, 3.2));
    Console.WriteLine(Add(9.8, 7));
    Console.WriteLine(Add("A", "B"));
    Console.WriteLine(Add("XYZ", 100));
    Console.ReadLine();
}
```

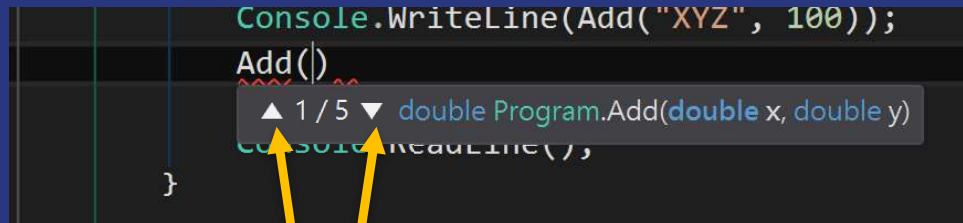
請將游標移到每個 Add 的呼叫上

```
static void Main(string[] args)
{
    Console.WriteLine(Add(1, 1));
    Console.WriteLine(Add(1, 2, 3));
    Console.WriteLine(Add(1.5, 3.2));
    Console.WriteLine(Add(9.8, 7));
    Console.WriteLine(Add("A", "B"));
    Console.WriteLine(Add("XYZ", 100));
    Console.ReadLine();
}
```

```
static void Main(string[] args)
{
    Console.WriteLine(Add(1, 1));
    Console.WriteLine(Add(1, 2, 3));
    Console.WriteLine(Add(1.5, 3.2));
    Console.WriteLine(Add(9.8, 7));
    Console.WriteLine(Add("A", "B"));
    Console.WriteLine(Add("XYZ", 100));
    Console.ReadLine();
}
```

在 Main Method 中隨便輸入個 Add()

```
Console.WriteLine(Add("XYZ", 100));  
Add()  
Console.ReadLine();  
}
```



操作一下這兩個三角形，你看到了甚麼？
執行前請將這個 Add() 刪除。

執行

討論

- 甚麼條件會讓方法稱為多載？
- 在寫程式的時候，如何判斷這個方法有多載？

